Feu tricolore intelligent avec capteur pressostats de Kambale kambere leonce Engineer

Code

// Feu Tricolore Intelligent 4 Voies - Capteurs Pressostats

// Comptage + Vitesse + Priorité Cortège + Piétons

// --- CONSTANTES TEMPS ---

const unsigned long T_VERT_MIN = 60000; // 60s

const unsigned long T_JAUNE_MIN = 20000; // 20s

const unsigned long T_ROUGE_MIN = 60000; // 60s

const unsigned long T_ADD_PAR_VOITURE = 3000; // +3s par voiture

// --- CONSTANTES DETECTION ---

const float DISTANCE_ENTRE_PRESSO = 2.0; // 2 mètres entre les 2 pressostats

const float VITESSE_CORTEGE_BASSE = 10.0; // km/h deuil/mariage

const float VITESSE_CORTEGE_HAUTE = 60.0; // km/h autorités

const unsigned long ANTI_REBOND = 500; // 500ms pour ne pas

```
compter 2x

// --- STRUCTURE VOIE ---
struct Voie {
  int pinV, pinJ, pinR; // LEDs Feu
  int pinP1, pinP2; // Pressostat 1 et 2

  int compteurVoitures;
  bool priorite;
  unsigned long tempsVert;

  // Variables internes
  bool etatP1, etatP2, dernierEtatP1, dernierEtatP2;
  unsigned long t1; // temps appui pressostat 1
  unsigned long dernierAppui;
};

Voie voies[4];
int voieActive = 0;
unsigned long chronoDepartVert;
```

```
// Pins Piétons
const int btnPieton1 = 2;
const int btnPieton2 = 3;
const int ledPieton1 = 46;
const int ledPieton2 = 47;

void setup() {
  Serial.begin(9600);

  // Init Voie A Route 1
  voies[0] = {22,23,24, 30,31, 0,false,T_VERT_MIN, 1,1,1,1, 0,0};
  // Init Voie B Route 1
  voies[1] = {25,26,27, 32,33, 0,false,T_VERT_MIN, 1,1,1,1, 0,0};
  // Init Voie C Route 2
  voies[2] = {28,29,30, 34,35, 0,false,T_VERT_MIN, 1,1,1, 0,0};
  // Init Voie D Route 2
  voies[3] = {31,32,33, 36,37, 0,false,T_VERT_MIN, 1,1,1,1, 0,0};

  // Init Pins
```

```
  for(int i=0; i<4; i++){
    pinMode(voies[i].pinV, OUTPUT);
    pinMode(voies[i].pinJ, OUTPUT);
    pinMode(voies[i].pinR, OUTPUT);
    pinMode(voies[i].pinP1, INPUT_PULLUP); // Pressostat = LOW
quand appuyé
    pinMode(voies[i].pinP2, INPUT_PULLUP);
  }

  pinMode(btnPieton1, INPUT_PULLUP);
  pinMode(btnPieton2, INPUT_PULLUP);
  pinMode(ledPieton1, OUTPUT);
  pinMode(ledPieton2, OUTPUT);

  toutRouge();
  delay(2000);
  passerAuVert(0);
}

void loop() {
```

```
  for(int i=0; i<4; i++){
    lirePressostats(i);
  }

  if(digitalRead(btnPieton1)==LOW ||
digitalRead(btnPieton2)==LOW){
    gererPietons();
  }

  gererCycle();
}

// --- LECTURE PRESSOSTATS ---
void lirePressostats(int i){
  voies[i].etatP1 = digitalRead(voies[i].pinP1);
  voies[i].etatP2 = digitalRead(voies[i].pinP2);

  // FRONT DESCENDANT sur Presso 1 = Voiture arrive
  if(voies[i].etatP1==LOW && voies[i].dernierEtatP1==HIGH){
    if(millis() - voies[i].dernierAppui > ANTI_REBOND){ // Anti-
```

```
rebond
      voies[i].compteurVoitures++;
      voies[i].t1 = millis();
      voies[i].dernierAppui = millis();
      Serial.print("Voie "); Serial.print(i); Serial.print(" +1 voiture.
Total:");
      Serial.println(voies[i].compteurVoitures);
    }
  }

  // FRONT DESCENDANT sur Presso 2 = Fin mesure vitesse
  if(voies[i].etatP2==LOW && voies[i].dernierEtatP2==HIGH){
    if(voies[i].t1 > 0){
      unsigned long t2 = millis();
      float tempsSec = (t2 - voies[i].t1) / 1000.0;
      float vitesseMS = DISTANCE_ENTRE_PRESSO / tempsSec;
      float vitesseKMH = vitesseMS * 3.6;

      Serial.print("Vitesse Voie "); Serial.print(i); Serial.print(": ");
      Serial.print(vitesseKMH); Serial.println(" km/h");
```

```
      // Détection Cortège
      if(vitesseKMH > 0 && (vitesseKMH < VITESSE_CORTEGE_BASSE || vitesseKMH > VITESSE_CORTEGE_HAUTE)){
        voies[i].priorite = true;
        Serial.print("!!! PRIORITE DETECTEE VOIE "); Serial.println(i);
      }
      voies[i].t1 = 0;
    }
  }

  voies[i].dernierEtatP1 = voies[i].etatP1;
  voies[i].dernierEtatP2 = voies[i].etatP2;
}

// --- GESTION CYCLE INTELLIGENT ---
void gererCycle(){
  // 1. Priorité Cortège
  for(int i=0; i<4; i++){
```

```
    if(voies[i].priorite && i!=voieActive){
      voies[i].priorite = false;
      passerAuVert(i);
      return;
    }
  }

  // 2. Temps adaptatif
  voies[voieActive].tempsVert = T_VERT_MIN +
(voies[voieActive].compteurVoitures * T_ADD_PAR_VOITURE);
  if(voies[voieActive].tempsVert > 120000)
voies[voieActive].tempsVert = 120000;

  if(millis() - chronoDepartVert > voies[voieActive].tempsVert){
    int prochaine = choisirProchaineVoie();
    passerAuVert(prochaine);
  }
}

void passerAuVert(int nouvelleVoie){
```

```
  setFeu(voieActive, 0,1,0); // Orange
  delay(T_JAUNE_MIN);
  setFeu(voieActive, 0,0,1); // Rouge
  delay(T_ROUGE_MIN); // Sécurité

  voieActive = nouvelleVoie;
  setFeu(voieActive, 1,0,0); // Vert
  voies[voieActive].compteurVoitures = 0; // Reset après passage
  chronoDepartVert = millis();
}

int choisirProchaineVoie(){
  int routeSuivante = (voieActive < 2)? 2 : 0;
  if(voies[routeSuivante].compteurVoitures >=
voies[routeSuivante+1].compteurVoitures)
    return routeSuivante;
  else
    return routeSuivante + 1;
}
```

```
void gererPietons(){
  Serial.println("Demande Pieton");
  setFeu(voieActive, 0,1,0);
  delay(T_JAUNE_MIN);
  setFeu(voieActive, 0,0,1);

  digitalWrite(ledPieton1, HIGH);
  digitalWrite(ledPieton2, HIGH);
  delay(15000);

  digitalWrite(ledPieton1, LOW);
  digitalWrite(ledPieton2, LOW);
}

// --- UTILS ---
void setFeu(int voie, bool v, bool j, bool r){
  digitalWrite(voies[voie].pinV, v);
  digitalWrite(voies[voie].pinJ, j);
  digitalWrite(voies[voie].pinR, r);
}
```

```
void toutRouge(){
  for(int i=0; i<4; i++) setFeu(i,0,0,1);
}
```